

Enhanced V-model

Mustafa S. Durmus¹, Ilker Ustoglu², Roman Y. Tsarev³ and Josef Börcsök⁴

¹ Electrical and Electronics Engineering Department, Pamukkale University, 20070, Denizli, Turkey (e-mail: msdurmus@pau.edu.tr).

² Department of Control and Automation Engineering, Yildiz Technical University, Istanbul, Turkey (e-mail: ustoglu@yildiz.edu.tr).

³ Informatics Department, Siberian Federal University, Krasnoyarsk, Russia (e-mail: tsarev.sfu@mail.ru).

⁴ Computer Architecture and System Programming Department, Kassel University, Kassel, Germany (e-mail: j.boercsoek@uni-kassel.de).

Abstract—Typically, software development processes are time consuming, expensive, and rigorous, particularly for safety-critical applications. Even if guidelines and recommendations are defined by sector-specific functional safety standards, development may not be completed because of excessive costs or insufficient planning. The V-model is one of the most well-known software development life cycle models. In this study, the V-model is modified by adding an intermediate step. The proposed modification provides three advantages: (1) it checks whether the constructed model covers all software requirements related to faults; (2) it enables simple coding; and (3) it decreases costs by early detection of modeling deficiencies before the model coding and testing phases. Since the proposed modification includes discrete event system-based fault diagnosis, its applicability to only automata and Petri net models (modules) can be considered a disadvantage.

Index Terms—Software development lifecycle, V-model, fault diagnosis, discrete event systems, EN 50128, fixed-block railway signaling systems.

1 INTRODUCTION

THE concept known as Safety Integrity Level (SIL) is used to quantify safety. The SIL is a degree of safety system performance for a Safety Instrumented System (SIS), which is an automatic system used to avoid accidents and to reduce their impact both on humans and the environment. An SIS has to execute one or more Safety Instrumented Functions (SIFs) to maintain a safe state for the equipment under control [1]. Bear in mind that, the safe state is known as the state where the whole system is prevented from falling into a dangerous situation. An SIF has a designated SIL level depending on the ratio of risk that needs to be decreased. IEC 61508, the standard for functional safety of electrical/electronic/programmable-electronic safety-related systems (SRSs), mentions that an SIL should be designated to each SIF and defines the safety integrity as the probability of a Safety Related System (SRS) adequately performing the required safety functions under all the stated conditions within a given period of time. From the lowest requirement level (SIL 1) to highest requirement level (SIL 4), SILs have four levels in the standards.

In IEC 61508, SRSs are classified into two types of operations: high and low demand. When the demand rate is both less than once a year and less than twice the proof test frequency, the SIS operates in a low demand mode. Otherwise, a high demand mode of operation is taken into account. Here, Probability of Failure on Demand (PFD) is the main reliability measure for low-demand SISs. On the other hand, for high demand operations the Probability of Failure per Hour (PFH) is the preferred measure. The SIL verifies whether the average probability of failure on demand (PFDavg) of the underlying SRS meets the required failure measure, where the average is computed over the operational time interval between periodic proof tests [2]. The target failure measures are tabulated in Tables 2 and 3 of IEC 61508-1 Edition 2 or in Tables 3 and 4 of IEC 61511-1 [3]. IEC 61511 has been developed as a process sector implementation of the generic standard IEC 61508.

When SRS are taken into consideration, the failure rate (λ) is one of the most important parameters. While, the failure rate of a new parameter can be established experimentally, for existing components it can be found in special handbooks. The reliability function, which describes the failures of items, is completely determined by λ .

The third part of IEC 61508 applies to any software used to develop a safety-related system within the scope of first and second parts of the standard, and establishes the requirements for safety life cycle phases. Industry and domain specific implementations of IEC 61508 include IEC 61511 for industrial processes, IEC 61513 for the nuclear industry, and IEC 62061 for machinery etc.

A lifecycle model is defined in [4] as a *model that describes stages in a software product development process*. The IEC 61508-4 standard discusses the term life cycle in the context of both *safety lifecycle* and *software lifecycle*. The safety life cycle includes the necessary activities involved in the implementation of SRSs [5]. IEC 61508 states that a safety life cycle for software development shall be selected and specified during the safety-planning phase in accordance with Clause 6 of IEC 61508-1. The safety life cycle includes the definition of scope, hazard and risk analysis, determination of safety requirements, installation, commissioning, validation, operation, maintenance, repair, and decommissioning. On the other hand, the software life cycle includes the activities occurring from the conception of the software to the decommissioning of the software. This life cycle deals with the definition of the software requirements, software design, software integration, and software tests.

Numerous life cycle models have been addressed in the literature, such as the waterfall, spiral, iterative development, and butterfly models [6–12]. However, despite the availability of many life cycle alternatives, railway-related safety standards such as IEC 61508, EN 50126, EN 50128, and IEC 62278 recommend using the V-model for software

development processes. The V-model life cycle has been applied to various domains such as the automotive [13], aerospace [14], railways [15], and the nuclear industry [16].

In this study, a Discrete Event System (DES)-based fault diagnosis method is added to the V-model life cycle as an intermediate step. The proposed modification provides three advantages: (1) it checks whether the constructed software model covers all software requirements related to faults; (2) it decreases costs by the early detection of modeling deficiencies prior to the V-model coding and testing phases; and (3) it enables simple coding. Since the proposed modification includes DES-based fault diagnosis, it is only applicable to automata and Petri net models (software modules), which can be considered a disadvantage.

Even if there are many model checking tools and techniques in the literature [17-21], the proposed enhancement is not complex as previously proposed methods. Because the diagnoser is built from the model itself and; since the modular approach is a must in the *Software Design* phase of the V-model in EN 50128 [22] (recommended as mandatory) there is no need to any tool to check the diagnosability of a simple module (component) model.

The remainder of this paper is organized as follows. The V-model lifecycle and the DES-based fault diagnosis scheme are explicated in Sections II and III, respectively. The enhanced V-model is introduced in Section IV, and conclusion section is given in Section V.

2 V-MODEL LIFECYCLE

Paul Rook introduced the V-model life cycle in 1986 as a guideline for software development processes [4]. The primary aim of the V-model is to improve both the efficiency of software development and the reliability of the produced software. The V-model offers a systematic roadmap from project initiation to product phase-out [4]. The V-model also defines the relationship between the development and test activities; it implements verification of each phase of the development process rather than testing at the end of the project. The V-model, as defined in IEC 61508-3, is shown in Fig. 1.

Before initializing a *software development process* according to the V-model, a *software planning phase* has to be realized, wherein a *software quality assurance plan*, *software verification* and *software validation plans*, and a *software maintenance plan* are fully defined. Later, the software requirements should be determined in cooperation with both the customer and the stakeholders. Using the selected software architectures (including modeling methods), software modules (or components) are developed by the designers. Each phase is verified immediately after completion. Note that the left side of the V-model (Fig. 1) represents the decomposition of the problem from the business world to the technical world [23]. After the coding phase, the right side of the V-model denotes the testing phase of the developed software.

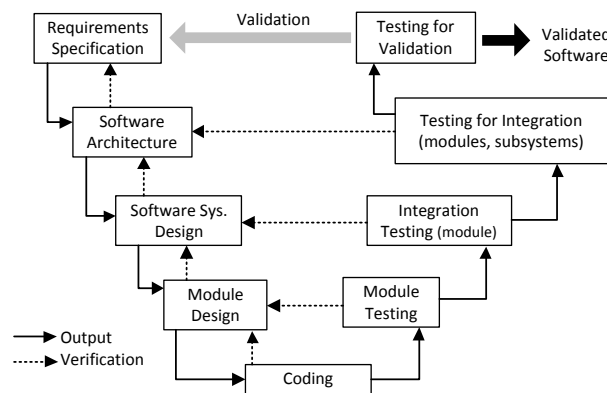


Fig. 1. V-model software safety integrity and development lifecycle [18].

The advantages and disadvantages of the V-model can be summarized as follows [6, 7, 23]:

Advantages

1. Facilitates greater control due to the standardization of products in the process.
2. Cost estimation is relatively easy due to the repeatability of the process.
3. Each phase has specific products.
4. Greater likelihood of success because of the early development of test plans and documentation before coding.
5. Provides a simple roadmap for the software development process.

Disadvantages

1. Low flexibility, expensive, and difficult to change scope.
2. No early prototypes.

3. Addresses software development within a project rather than a whole organization.
4. Too simple to precisely reflect the software development process and may steer managers into a false sense of security.

3 DES-BASED FAULT DIAGNOSIS

An *event* is defined as an encountered specific action, i.e., an unplanned incident that occurred naturally or due to numerous conditions that are encountered simultaneously [25]. A DES is a discrete-state, event-driven system in which the state evolution of the system depends totally on the occurrence of discrete events over time. Events are classified as observable or unobservable events in a DES.

In DES-based fault diagnosis, diagnosability can be summarized as follows. A system (or software module) is considered as diagnosable if it is possible to identify, within a finite delay, occurrences of precise unobservable events that are referred to as fault events [26]. In other words, a system is accepted as diagnosable if the fault type is always identified within a uniformly bounded number of transition firings after the occurrence of the fault [27]. The diagnoser is obtained from the system model itself and carries out diagnostics to observe the system behavior. Diagnoser states involve fault information, and occurrences of faults are identified within a finite delay by examining these states [28]. Although the meanings of *failure* and *fault* are considered the same by DES researchers, they are defined differently in the EN 50128 safety standard. In this study, we use the term *fault* rather than *failure*.

Finite state machines and Petri nets are considered as DES-based modeling methods and, these methods are also highly recommended by railway-related functional safety standards [22]. Therefore, DES-based fault diagnosis can be used when developing software modules for railway signaling software.

3.1 Basic Petri net (PN) Definitions

A Petri net [29] is defined as;

$$PN = (P, T, F, W, M_0) \quad (1)$$

where

- $P = \{p_1, p_2, \dots, p_k\}$ is the finite set of places,
- $T = \{t_1, t_2, \dots, t_n\}$ is the finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs,
- $W: F \rightarrow \{1, 2, 3, \dots\}$ is the weight function,
- $M_0: P \rightarrow \{0, 1, 2, 3, \dots\}$ is the initial marking, and
- $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

The sets $O(t_j)$ and $I(t_j)$ are used to denote output places and input places of transition t_j , respectively, as follows:

$$O(t_j) = \{p_i \in P : (t_j, p_i) \in F\}, \quad (2)$$

$$I(t_j) = \{p_i \in P : (p_i, t_j) \in F\} \quad (3)$$

For a marking M , $M(p_i) = n$ represents the token number of the i th place where it is equal to n [29]. Representation of a marking $M: P \rightarrow \{1, 2, 3, \dots\}$ can be realized by a k -element vector, where k denotes the total number of places.

Definition 1 [25]: A transition t_j assumed as *ready to fire* (enabled) at a marking M if each input place p_i of t_j has at least $W(p_i, t_j)$ tokens, where $W(p_i, t_j)$ is the arc weight from place p_i to transition t_j , i.e., $M(p_i) \geq W(p_i, t_j)$ for all $p_i \in I(t_j)$.

Keep in mind that the transition t_j is always ready to fire if $I(t_j) = \emptyset$. Firing of an enabled transition depends on whether the event actually occurs or not. When an enabled transition t_j is fired, $W(p_i, t_j)$ tokens are removed from each input place $p_i \in I(t_j)$ and $W(t_j, p_i)$ tokens are added to each output place $p_i \in O(t_j)$, where $W(t_j, p_i)$ is the weight of the arc from t_j to p_i . Thus,

$$M'(p_i) = M(p_i) - W(p_i, t_j) + W(t_j, p_i), \quad (4)$$

Here, the new token number in the i th place is represented by $M'(p_i)$ after the firing of transition t_j . If a transition t_j is enabled at a marking M , then it is represented by $M[t_j > .$ In addition, if the firing of t_j at M results with

M' , then it is represented by $M[t_j > M']$.

Definition 2 [29]: If a PN has no self-loops, then it is considered as *pure* and when all arc weights of a PN are 1, the it is said to be *ordinary*.

Definition 3 [29]: $M_0[t_1 > M_1[t_2 > \dots M_{k-1}[t_k > M_k]$ means that, the marking M_k is reachable from the initial marking M_0 by the $t_1 t_2 \dots t_k$ transitions sequence. Note that, $R(M_0)$ denotes the set of all reachable markings from M_0 .

Definition 4 [29]: A PN is said to be l -bounded if the token number in each place do not exceed a finite number l , i.e., $\forall M_k \in R(M_0), \forall p_i \in P: M_k(p_i) \leq l$. Note that a PN is 1-bounded then, it is assumed as *safe*.

Definition 5 [29, 30]: A PN is said to be *free from deadlocks* if it is possible to find at least one enabled transition at every reachable marking, $M_k \in R(M_0)$.

The set of places, P is partitioned into a set of observable places and a set of unobservable places (P_o and P_{uo}) [27]. Likewise, the set of transitions, T is partitioned into a set of observable transitions and a set of unobservable transitions (T_o and T_{uo}). Thus, the partitioned sets for P and T can be expressed as

$$P = P_{uo} \cup P_o \text{ and } P_{uo} \cap P_o = \emptyset, \quad (5)$$

$$T = T_{uo} \cup T_o \text{ and } T_{uo} \cap T_o = \emptyset. \quad (6)$$

In addition, a subset T_F of T_{uo} represents a faulty transitions set. It is assumed that there are m different fault types. Here, $\Delta_F = \{F_1, F_2, \dots, F_m\}$ is the set of fault types. T_F is expressed as

$$T_F = T_{F_1} \cup T_{F_2} \cup \dots \cup T_{F_m}, \quad (7)$$

where $T_{F_i} \cap T_{F_j} = \emptyset$ (if $i \neq j$). $\Delta = \{N\} \cup 2^{\Delta_F}$ is used to define the label set, where N is used to represent the label “normal,” which specifies that all fired transitions are *not faulty*, and 2^{Δ_F} represents the power set of Δ_F , i.e., 2^{Δ_F} is the set of all subsets of Δ_F . In the remainder of this paper, unobservable transitions and places are represented as shown in Fig. 2.

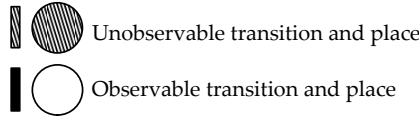


Fig. 2. Unobservable and observable places and transitions.

3.2 Diagnosis of Faults Using PN Models

Since the system model contains unobservable places, it is not always possible to distinguish some markings. Thus, if $M_1(p_i) = M_2(p_i)$ for any $p_i \in P_o$, then its denotes as $M_1 \equiv M_2$. That is to say, M_1 and M_2 markings have the same observations. As done in [31], the definition of the quotient set $\hat{R}(M_0)$ according to the equivalence relation (\equiv) is useful; $\hat{R}(M_0) := R(M_0) / \equiv := \{\hat{M}_0, \dots, \hat{M}_n, \dots\}$ where $M_0 \in \hat{M}_0$. An observable marking or the observation of a marking is represented by each member of $\hat{R}(M_0)$. We assume the following two statements are true for simplicity.

Assumption 1 [26, 27]: A PN given by equation (1) is *free from deadlocks*.

Assumption 2 [26, 27]: There does not exist an order of unobservable transitions whose firing produces a cycle of markings that have the same observation; i.e., for any $M_i \in R(M_0)$ and $t_k \in T_{uo}, k=1, 2, \dots, n$, $M_1[t_1 > M_2[t_2 > \dots M_k[t_k > M_1]$ $k, j=1, 2, \dots, n: M_k \neq M_j$.

At this point, a diagnoser definition [26, 27, 31] is given for a PN. A diagnoser state q_d is given as $q_d = \{(M_1, l_1), (M_2, l_2), \dots, (M_n, l_n)\}$, which involves pairs of a marking $M_i \in R(M_0)$ and a label $l_i \in \Delta$. The symbol $Q = 2^{R(M_0) \times \Delta}$ represents the power set of $R(M_0) \times \Delta$; i.e., each member of Q is a subset of $R(M_0) \times \Delta$ and is given as $\{(M_1, l_1), (M_2, l_2), \dots, (M_n, l_n)\}$. The diagnoser is given by

$$G_d = (Q_d, \Sigma_o, \delta_d, q_0). \quad (8)$$

The diagnoser given by (8) is an automaton where the set of states are represented by $Q_d \subseteq Q$, the set of events are represented by $\Sigma_o = \hat{R}(M_0) \cup T_o$, the notation $\delta_d: Q_d \times \Sigma_o \rightarrow Q_d$ represents the partial state transition function, and the initial state is denoted by $q_0 = \{(M_0, N)\}$. The state set $Q_d \subseteq Q$ represents the reachable states from the initial state q_0 by using δ_d . Each observed event $\sigma_o \in \Sigma_o$ represents an observation of a marking in $\hat{R}(M_0)$ or an observable transition in T_o . The state transition function δ_d is defined with the use of the label propagation function and the range function. The label propagation function $LP: R(M_0) \times \Delta \times T^* \rightarrow \Delta$ transfers the label (faulty or normal) over an order of transitions $s \in T^*$. Here, T^* denotes the set of all finite sequences of the members of T [26, 27]:

$$LP(M, l, s) = \begin{cases} N, & \text{if } (l = N) \wedge (\forall F_i \in \Delta_F : T_{F_i} \notin s), \\ \{F_i : F_i \in l \vee T_{F_i} \in s\}, & \text{otherwise,} \end{cases} \quad (9)$$

where $T_{F_i} \in s$ ($T_{F_i} \notin s$) indicates that a sequence of transitions $s \in T^*$ includes (or does not include) a faulty transition with fault type F_i . If the sequence of transitions does not contain any faulty transition, in this case, the resulting marking is labeled as normal (N). If the sequence of transitions contains a faulty transition, then the resulting marking is labeled with the corresponding fault type. In that case, the range function $LR: Q \times \Sigma_o \rightarrow Q$ is determined as follows:

$$LR(q, \sigma_o) = \bigcup_{(M, l, q \in T^* \setminus \{M, \sigma_o\})} M, LP(M, l, s) \}, \quad (10)$$

where $M[s > M']$ and $T^*(M, \sigma_o) \subseteq T^*$ is defined in the next two cases.

1. If $\sigma_o \in \hat{R}(M_0)$,

$$T^*(M, \sigma_o) = \begin{cases} \emptyset, & \text{if } M \in \sigma_o, \\ \left\{ s \in T_{uo}^* : \begin{aligned} & (M_s \in \sigma_o) \\ & \wedge (\forall s' (\neq s) \in \bar{s} : M_{s'} \equiv M) \end{aligned} \right\}, & \text{otherwise,} \end{cases} \quad (11)$$

where $M[s > M_s]$, $M[s' > M_{s'}]$, and \bar{s} denotes the set of all prefixes of s . In (11), the case of $M \notin \sigma_o$ corresponds to a change of the observable marking. Here, $T^*(M, \sigma_o)$ is the set of sequences $s \in T_{uo}^*$ of unobservable transitions such that during the firing of s , all interval observable markings except the last one in σ_o are the same.

2. If $\sigma_o \in T_o$, [27]

$$T^*(M, \sigma_o) = \left\{ s \in T_{uo}^* \cdot \{\sigma_o\} : \begin{aligned} & (M[s >]) \wedge (\forall s' (\neq s) \in \bar{s} : M_{s'} \equiv M) \end{aligned} \right\}, \quad (12)$$

where $M[s' > M_{s'}]$. If the firing of an observable transition $\sigma_o \in T_o$ is observed, then $T^*(M, \sigma_o)$ is the set of sequences of unobservable transitions followed by σ_o such that all interval observable markings except the last one are the same. In other words, $T^*(M, \sigma_o)$ is the set of possible transition sequences from M that are consistent with the observed event σ_o .

Lastly, the state transition function $\delta_d: Q_d \times \Sigma_o \rightarrow Q_d$ is defined as follows [26, 27]:

$$\delta_d(q, \sigma_o) = \begin{cases} LR(q, \sigma_o), & \text{if } LR(q, \sigma_o) \neq \emptyset \\ \text{undefined,} & \text{otherwise.} \end{cases} \quad (13).$$

3.3 Obtaining Diagnosability

A PN is accepted as diagnosable, if the faulty type is identified for all time within a uniformly bounded number of transition firings after the fault occurrence [27]. In other words, a PN is considered as diagnosable if, and only if, the diagnoser given by (8) does not involve an F_m -indeterminate cycle for any fault type F_m . Detailed explanation and the proof of this theorem is given in [26].

3.4 Railway Point Example

Trains can pass from one track to another by railway *points* placed at necessary locations. Since the trains do not have any steering mechanism, they use railway points to pass from one track to another. Points have two position indications, i.e., Normal (*Nr*) and Reverse (*Rev*). At any railway point, three main faults may occur. These faults are identified in the V-model software requirements specification phase as follows:

- F_1 : Point may not reach the desired position in a predefined time (e.g., 5 s) while moving from *Nr* to *Rev*.
- F_2 : Point may not reach the desired position in a predefined time (e.g., 5 s) while moving from *Rev* to *Nr*.
- F_3 : Both position indications may be received simultaneously.

Examples of diagnosable and non-diagnosable *PN* models of a railway point are given in Fig. 3 and Fig. 4, respectively. The meanings of the transitions and places of the models in Fig. 3 and Fig. 4 are given in Table I and Table II, respectively. Note that the striped places and transitions represent unobservable places (P_{uo}) and transitions (T_{uo}), whereas the other places (P_o) and transitions (T_o) are observable. M_0 represents the initial marking of the Petri net.

The representation of the *PN* model in Fig. 3 is as follows:

$$\begin{aligned}
 P_o &= \{P_{PM_1}, P_{PM_2}, P_{PM_5}, P_{PM_6}, P_{PM_8}, P_{PM_10}, P_{PM_12}\}, \\
 P_{uo} &= \{P_{PM_3}, P_{PM_4}, P_{PM_7}, P_{PM_9}, P_{PM_11}\}, \\
 T_o &= \{t_{PM_1}, t_{PM_2}, t_{PM_3}, t_{PM_4}, t_{PM_5}, t_{PM_6}, t_{PM_7}, t_{PM_8}, t_{PM_9}, \dots \\
 &\quad \dots t_{PM_10}, t_{PM_11}, t_{PM_12}, t_{PM_13}, t_{PM_14}\}, \\
 T_{uo} &= \{t_{PM_f1}, t_{PM_f2}, t_{PM_f3}\}, \\
 M_0 &= (M_0(P_{PM_1}), M_0(P_{PM_2}), M_0(P_{PM_3}), M_0(P_{PM_4}), \dots \\
 &\quad \dots M_0(P_{PM_5}), M_0(P_{PM_6}), M_0(P_{PM_7}), M_0(P_{PM_8}), \dots \\
 &\quad \dots M_0(P_{PM_9}), M_0(P_{PM_10}), M_0(P_{PM_11}), M_0(P_{PM_12})) \\
 &= (0, 0, \underline{0}, \underline{0}, 1, 0, \underline{0}, \underline{0}, \underline{0}, \underline{0}, \underline{1}, 0).
 \end{aligned} \tag{14}$$

The three different fault types given in Fig. 3 are $\Delta_F = \{F_1, F_2, F_3\}$, where $T_{F_1} = \{t_{PM_f1}\}$, $T_{F_2} = \{t_{PM_f2}\}$, and $T_{F_3} = \{t_{PM_f3}\}$. The rectangles are used to diminish the complexity of the *PN* model. Each rectangle represents the label of the related place.

The diagnoser illustrated in Fig. 3 is built from the railway point model itself. A rectangle is used to denote each state and each state contains a pair of place markings and an attached label, normal (*N*) or fault ($F_m, m \in \{1, 2, 3\}$). In other words, in parts of the diagnoser, a marking immediately after an observed event is detected precisely.

In accordance with the definition of the diagnoser in (8), a label which represents an observable transition or the observation of a marking is attached to all diagnoser state transitions. In this study, with a slight abuse of notation, labels containing the observation of a marking or a pair of the observation of a marking and an observable transition are attached to all state transitions of the diagnoser.

For example, at \hat{M}_{PM_0} in Fig. 3, the event label \hat{M}_{PM_10} represents that the observable marking \hat{M}_{PM_10} is observed by firing the unobservable transition t_{PM_f3} . Similarly, the diagnoser state changes from $\{((0, 0, \underline{0}, \underline{0}, 1, 0, \underline{0}, \underline{0}, \underline{0}, \underline{0}, \underline{1}, 0), N)\}$, to $\{((0, 1, \underline{0}, \underline{0}, 1, 0, \underline{0}, \underline{0}, \underline{0}, \underline{0}, \underline{1}, 0), N)\}$, as a function of firing the observable transition t_{PM_2} with the observation \hat{M}_{PM_1} of the resulting marking. According to the definition given in Section III-C, since there is no F_i -indeterminate cycle in the diagnoser, the *PN* model is diagnosable.

TABLE 1
DEFINITIONS OF TRANSITIONS AND PLACES IN THE MODELS GIVEN IN FIG. 3.

Place	Definition	Transition	Definition
P_{PM_1}	Nr position requested	t_{PM_1}	Safety criteria are met, point Nr position request
P_{PM_2}	Rev position requested	t_{PM_2}	Safety criteria are met, point Rev position request
P_{PM_3}	Point is moving to Nr position	t_{PM_3} (t_{PM_6})	Request ignored
P_{PM_4}	Point is moving to Rev position	t_{PM_4}	Point left the Rev position
P_{PM_5}	Point is in Nr position	t_{PM_5}	Point left the Nr position
P_{PM_6}	Point is in Rev position	t_{PM_7} (t_{PM_8})	Point reached to Nr (Rev) position
P_{PM_7}	Fault type F_1 has occurred	t_{PM_9} (t_{PM_10})	Predefined filter time has expired
P_{PM_8}	Point is faulty (F_1)	t_{PM_11} (t_{PM_12})	Nr (Rev) position request
P_{PM_9}	Fault type F_2 has occurred	t_{PM_13} (t_{PM_14})	Point moved to Nr (Rev) position and the fault acknowledged
P_{PM_10}	Point is faulty (F_2)	t_{PM_f1}	Point indication fault
P_{PM_11}	Unobservable fault restriction	t_{PM_f2}	Point indication fault
P_{PM_12}	Point is faulty (F_3)	t_{PM_f3}	Point position fault

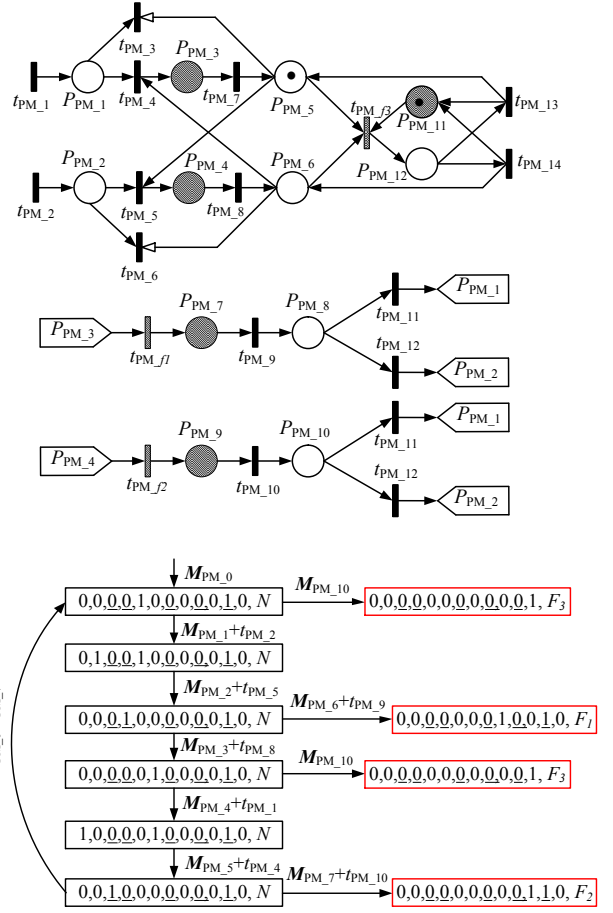


Fig. 3. PN model of a railway point and its diagnoser (diagnosable).

Representation of the PN model in Fig. 4 is as follows:

$$\begin{aligned}
 P_o &= \{P_{PM_1}, P_{PM_2}, P_{PM_3}, P_{PM_4}, P_{PM_5}, P_{PM_6}\}, \\
 P_{uo} &= \{P_{PM_7}\}, \\
 T_o &= \{t_{PM_1}, t_{PM_2}, t_{PM_3}, t_{PM_4}, t_{PM_5}, t_{PM_6}, t_{PM_7}, t_{PM_8}\}, \\
 T_{uo} &= \{t_{PM_f1}, t_{PM_f2}, t_{PM_f3}\}, \\
 M_0 &= (M_0(P_{PM_1}), M_0(P_{PM_2}), M_0(P_{PM_3}), M_0(P_{PM_4}), \dots \\
 &\quad \dots M_0(P_{PM_5}), M_0(P_{PM_6}), M_0(P_{PM_7})) \\
 &= (0, 0, 1, 0, 0, 0, 0).
 \end{aligned} \tag{15}$$

The diagnoser in Fig. 4 is not diagnosable because it is not possible to distinguish the fault type after observing the marking \hat{M}_{PM_5} . The software model in Fig. 4 seems to contain all the software requirements related to the faults; however, the developed software model is not diagnosable, which means that the PN model will identify only one of the faults, F_1 or F_2 . Therefore, the designers should revise the PN model before proceeding to the coding phase; otherwise, this deficiency will result in an unsuccessful test case in the V-model module-testing phase.

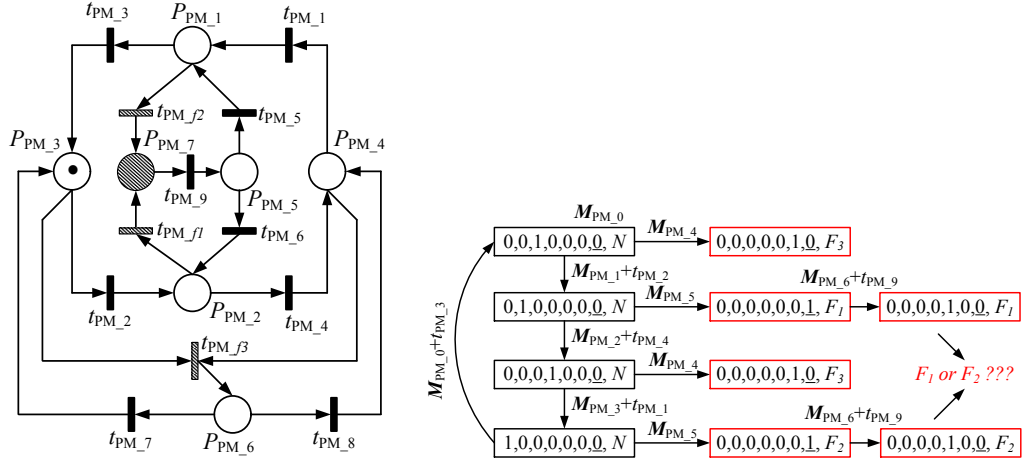


Fig. 4. PN model of a railway point and its diagnoser (not diagnosable).

4 MODIFIED V-MODEL LIFECYCLE

Here, the modification of the V-model by adding the DES-based fault diagnosis scheme is explained. As mentioned in [4], [33], and [34], the required workforce and the cost of the development process of the software increases towards the end with respect to the initial phases of the development life cycle. The proposed modification to the V-model enables designers to check their software modules one more time before proceeding to the coding phase. This additional control is realized by checking the diagnosability of each module.

As mentioned in Section III, if a module is diagnosable, then it fully meets the software requirements, particularly for requirements related to failure modes. In the usual software development process, the fulfillment of the requirements are checked by realizing the module tests. Each requirement should be tested at least once. Diagnosability analysis allows us to check software requirement – software module compatibility before obtaining the software source code.

This intermediate phase can be considered as time-consuming and an increase in workload. However, rather than turning back from the module testing phase to the module design phase, the proposed phase provides a final inspection of modules before proceeding to the coding and module testing phases. The proposed enhanced V-model is shown in Fig. 5.

TABLE 2
DEFINITIONS OF TRANSITIONS AND PLACES IN THE MODELS GIVEN IN FIG. 4.

Place	Definition	Transition	Definition
P_{PM_1}	Point is moving to Nr position	t_{PM_1} (t_{PM_2})	Movement request is received and safety criteria are met for Nr (Rev) position
P_{PM_2}	Point is moving to Rev position	t_{PM_3} (t_{PM_4})	Point reached to Nr (Rev) position
P_{PM_3}	Point position is Nr	t_{PM_5} (t_{PM_6})	Point request to Nr (Rev) position
P_{PM_4}	Point position is Rev	t_{PM_7} (t_{PM_8})	Point moved to Nr (Rev) position and the fault acknowledged
P_{PM_5}	Fault type F_1 or F_2 has occurred	t_{PM_9}	Predefined filter time has expired
P_{PM_6}	Point is faulty (F_3)	$t_{PM_{f1}}$ ($t_{PM_{f2}}$)	Point indication fault
P_{PM_7}	Point is moving from one position to another	$t_{PM_{f3}}$	Point position fault

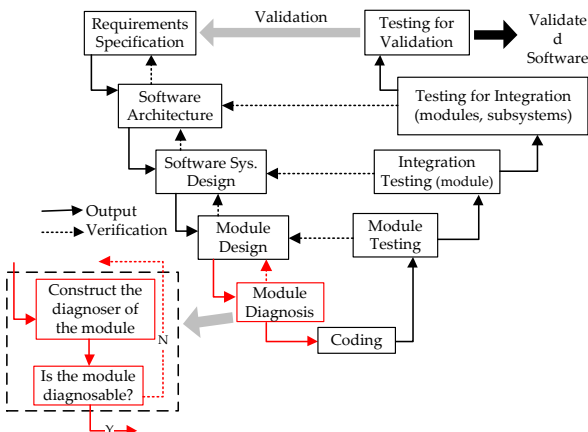


Fig. 5. Enhanced V-model (Y-Yes, N-No).

The proposed modification (Fig. 5) has three unique advantages.

1. It checks whether the constructed model covers all software requirements related to faults.

If the developed software model is not diagnosable, then the software model does not contain all software requirements.

2. It decreases costs through early detection of modeling deficiencies before proceeding to V-model coding and testing phases.

As can be seen in Fig. 5, after proceeding to the coding phase, the designer can only go back to the module design phase at the end of the module tests. The cost of fixing an error at the design phase is 3–8 units, whereas the cost of fixing an error at the testing phase is 21–78 units [34–36]. Another study showed that, it is 5 times more expensive to fix a problem at the design stage than in the course of initial requirements, 10 times more expensive to fix it through the coding phase, 20 to 50 times more expensive to fix it at acceptance testing and, 100 to 200 times more expensive to fix that error in the course of actual operation [37].

3. It enables designers to write simple and more readable code.

This is explained with a case study. An example PN model of a Two-Aspect Signal (TAS) and its diagnoser is given in Fig. 6. The notations of the transitions and places of the models in Fig. 6 are shown in Table III.

It is assumed that two faults may occur in a TAS. These faults are identified as follows:

- F_1 : Both signal aspects are lit at the same time;
- F_2 : No signals are lit.

A detailed example can be seen in [38].

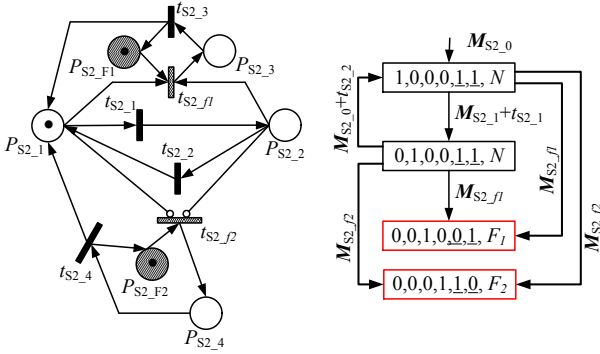


Fig. 6. TAS PN model and its diagnoser.

The Programmable Logic Controller (PLC) code snippet of the TAS model with and without a diagnoser is shown in Fig. 7 and Fig. 8, respectively. As can be seen in Fig. 7(a), the model with the diagnoser is simpler.

For the TAS PLC code given in Fig. 7(a),(b), the diagnoser compares the actual states of the PN model with its faulty states. When the faulty state of the diagnoser is fully matched with the actual PN states, the diagnoser sets the corresponding output to logic 1. This is illustrated in Fig. 9.

TABLE 3
DEFINITIONS OF TRANSITIONS AND PLACES IN THE MODELS GIVEN IN FIG. 6.

Place	Definition	Transition	Definition
P_{S2_1}	Signal is red	t_{S2_1}	Turn signal to green
P_{S2_2}	Signal is green	t_{S2_2}	Turn signal to red
P_{S2_3}	Fault type F_1 has occurred	t_{S2_3}	Signal turned to red and the fault acknowledged
P_{S2_4}	Fault type F_2 has occurred	t_{S2_4}	Signal turned to red and the fault acknowledged
P_{S2_F1}	Unobservable fault restriction	t_{S2_f1}	Point aspect fault
P_{S2_F2}	Unobservable fault restriction	t_{S2_f2}	Point indication fault

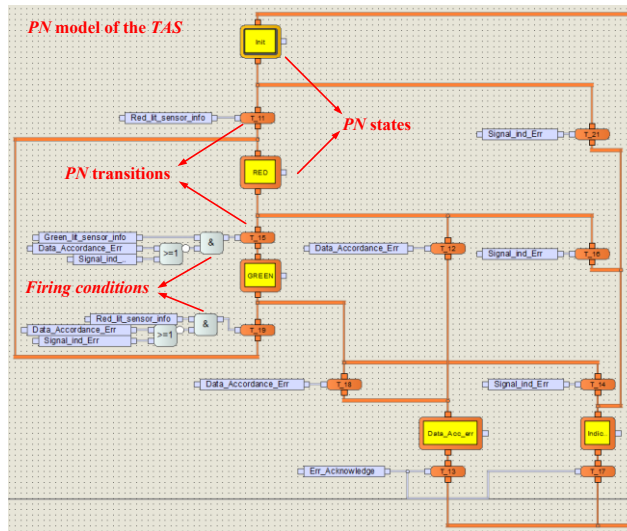


Fig. 7(a). TAS model with diagnoser..

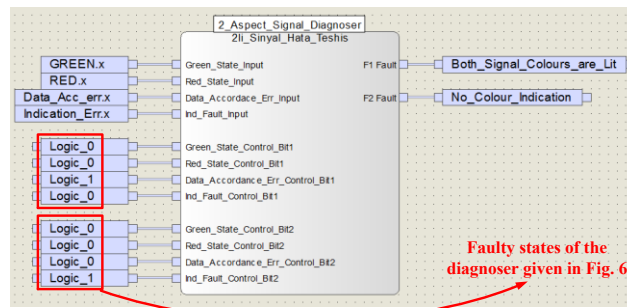


Fig. 7(b). The diagnoser block of the TAS.

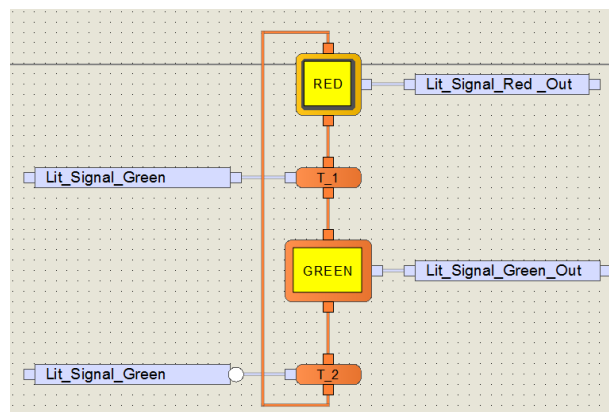


Fig. 8(a). TAS model without diagnoser.

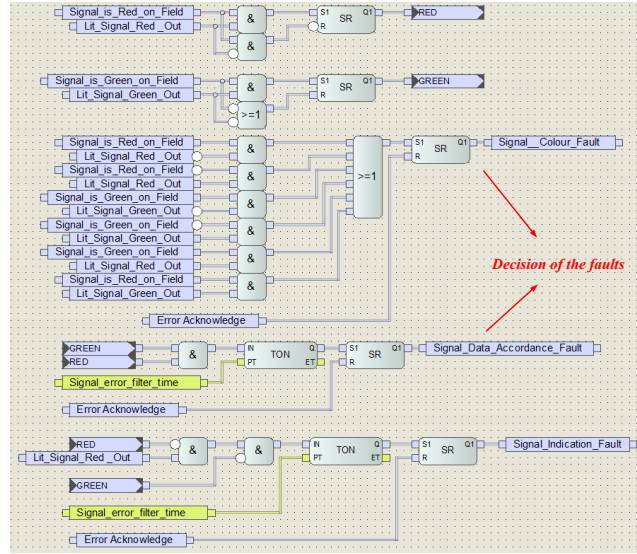


Fig. 8(b). Decision of the faults without diagnoser.

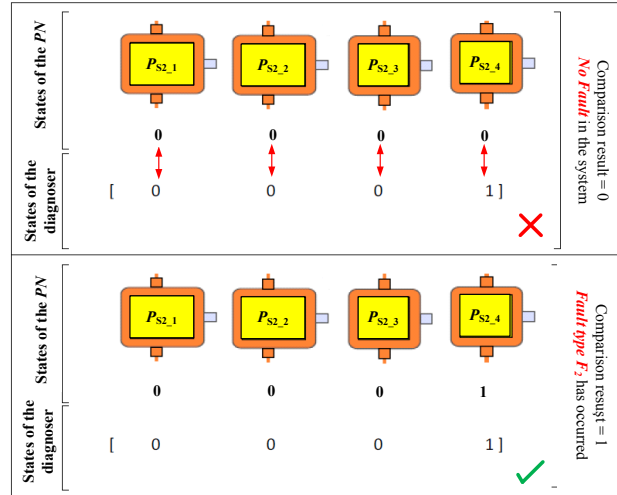


Fig. 9. Decision of the diagnoser given in Fig. 7(a),(b).

Moreover, in the illustration of the preferred organizational structure of the railway-related functional safety standard EN 50128, the requirement manager, the designer, and the implementer can be the same person for all safety integrity levels [22]. The preferred organizational structure of the enhanced V-model is given in Fig. 10.

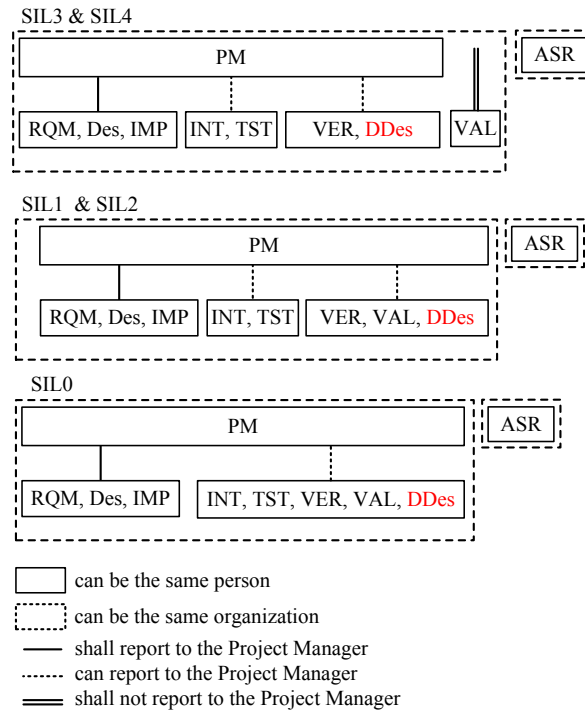


Fig. 10. Preferred organizational structure for the enhanced V-model (PM: Project Manager, RQM: Requirement Manager, Des: Designer, IMP: Implementer, VER: Verifier, VAL: Validator, DDes: Diagnoser Designer, ASR: Assessor).

5. CONCLUSION

Faults in a safety-critical system (or its subsystems) may cause severe harm to humans (e.g., railway, aircraft, and nuclear power station control systems). Therefore, the development steps of software for such safety-critical systems must be executed very carefully. Designers, developers, and engineers must consider the recommendations of both the international safety standards and the national rules to satisfy the required safety level and fulfill requirements. Developing software for such systems is guided by several software development life cycles, such as the well-known V-model.

Although enhancing the V-model with DES-based fault diagnosis is time consuming, however, the advantages of this intermediate step are threefold: (1) it checks whether the developed model fulfills all software requirements, especially those related to the failure mode; (2) the code developed with a diagnoser is simpler than the code produced without a diagnoser; and (3) an early check of the models is possible before proceeding to the testing phase (right side of the V-model) because the V-model leads developers from the module testing phase to the module design phase rather than the coding phase.

Moreover, when costs and work hours are considered, adding such an intermediate step to the V-model can result in considerable benefits to both project management and product development departments.

ACKNOWLEDGMENT

Corresponding author: Mustafa S. Durmus.

This work was supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) with the project number 115E394 - Fail-Safe PLC Implementation of Interlocking System Design with Fault Diagnosis Capability for Fixed-block Railway Signaling Systems.

The authors are thankful to Enago (www.enago.com) for the review of the English language of the paper.

REFERENCES

- [1] IEC61508, *Functional safety of electrical/ electronic/programmable electronic safety-related systems, Parts 1–7*. International Electrotechnical Commission, May 2010.
- [2] W. M. Goble, *Control System Safety Evaluation and Reliability*. 3rd ed. ISA, NC:Raleigh, 1998.
- [3] A. G. King, "SIL determination: Recognising and handling high demand mode scenarios," *Process Safety and Environmental Protection*, vol. 92, no. 4, pp. 324–328, 2014.
- [4] P. Rook, "Controlling Software Projects," *Software Engineering Journal*, vol. 1, no. 1, pp. 7–16, 1986.
- [5] IEC 61508-4, *Functional safety of electrical/electronic/programmable electronic safety-related systems, Part 4: Definitions and Abbreviations*, May 2010.

- [6] N. M. Munassar and A. Govardhan, "A Comparison Between Five Models of Software Engineering," *International Journal of Computer Science Issues*, vol. 7, no. 5, pp. 94-101, 2010.
- [7] S. T. Krishna, S. Sreekanth, K. Perumal and K. R. Kumar Reddy, "Explore 10 Different Types of Software Development Process Models," *International Journal of Computer Science and Information Technologies*, vol. 3, no. 4, pp. 4580-4584, 2012.
- [8] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," *Proc. Wescon*, pp. 1-9, 1970.
- [9] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, vol. 21, no. 5, pp. 61-72, 1988.
- [10] S. D. Morton, "The Butterfly Model for Test Development. Applied Dynamics International," https://www.agileconnection.com/sites/default/files/article/file/2014/The%20Butterfly%20Model%20for%20Test%20Development%20-%20XUS441382file1_0.pdf. 2001.
- [11] S. Easterbrook, "Lecture 4: Software Lifecycles the Butterfly Model for Test Development," University of Toronto, <http://www.cs.toronto.edu/~sme/CSC444F/slides/L04-Lifecycles.pdf>. 2001.
- [12] M. M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060-1076, 1980.
- [13] R. A. Rahman, U. Pulm and R. Stetter, "Systematic Mechatronic Design of a Piezo-Electric Brake," *International Conference on Engineering Design*, pp. 1-12, 2007.
- [14] L. Märtin, M. Schatalov, M. Hagner, U. Goltz and O. Maibaum, "A Methodology for Model-Based Development and Automated Verification of Software for Aerospace Systems," *IEEE Aerospace Conference*, pp. 1-19, 2013.
- [15] F. Scippacercola, R. Pietrantuono, S. Russo and A. Zentai, "Model-Driven Engineering of a Railway Interlocking System," *3rd International Conference on Model-Driven Engineering and Software Development*, pp. 509-519, 2013.
- [16] IAEA Safety Standards Series SSG-39, *Design of Instrumentation and Control Systems for Nuclear Power Plants*, 2016.
- [17] M. Kwiatkowska, G. Norman, D. Parker, "PRISM: Probabilistic Symbolic Model Checker," In: Field T., Harrison P.G., Bradley J., Harder U. (eds) *Computer Performance Evaluation: Modelling Techniques and Tools, TOOLS 2002. Lecture Notes in Computer Science*, vol. 2324, Springer, Berlin, Heidelberg, 2002.
- [18] G. Gardey, D. Lime, M. Magnin, O. Roux, "Romeo: A Tool for Analyzing Time Petri Nets," In: Etessami K., Rajamani S.K. (eds) *Computer Aided Verification, CAV 2005. Lecture Notes in Computer Science*, vol. 3576, Springer, Berlin, Heidelberg, 2005.
- [19] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," In: Brinksma E., Larsen K.G. (eds) *Computer Aided Verification, CAV 2002. Lecture Notes in Computer Science*, vol. 2404, Springer, Berlin, Heidelberg, 2002.
- [20] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, S. Yovine, "Kronos: A model-checking tool for real-time systems," In: Ravn A.P., Rischel H. (eds) *Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 1998. Lecture Notes in Computer Science*, vol. 1486, Springer, Berlin, Heidelberg, 1998.
- [21] G. Holzmann, *Spin model checker, the: primer and reference manual*. 1st Edition, Addison-Wisley, 2003.
- [22] BS EN 50128, *Railway Applications-Communication, Signalling and processing systems-Software for railway control and protection systems*, June 2011.
- [23] A. Ratcliffe, "SAS Software Development with the V-Model," *3SAS Global Forum 2011, Coder's Corner*, pp. 1-9, 2011.
- [24] IEC 61508-3, *Functional safety of electrical/electronic/programmable electronic safety-related systems, Part 3: Software Requirements*, May 2010.
- [25] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. 2nd Edition, Springer, 2008.
- [26] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1555-1575, 1995.
- [27] T. Ushio, I. Onishi and K. Okuda, "Fault detection based on Petri net models with faulty behaviours," *International Conference on Systems, Man, and Cybernetics*, pp. 113-118, 1998.
- [28] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen and D. Teneketzis, "Failure diagnosis using discrete-event models," *IEEE Transactions on Control Systems Technology*, vol. 4, no. 2, pp. 105-124, 1996.
- [29] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580, 1989.
- [30] Z. W. Li, M. C. Zhou and N. Q. Wu, "A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 2, pp. 173-188, 2008.
- [31] S. L. Chung, "Diagnosing PN-based models with partial observable transitions," *International Journal of Computer Integrated Manufacturing*, vol. 18, no. 2-3, pp. 158-169, 2005.
- [32] M. S. Durmuş, S. Takai and M. T. Söylemez, "Fault Diagnosis in Fixed-Block Railway Signaling Systems: A Discrete Event Systems Approach," *IEEE Transactions on Electrical and Electronic Engineering*, vol. 9, pp. 523-531, 2014.
- [33] B. W. Boehm, "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software*, vol. 1, no. 1, pp. 75-88, 1984.
- [34] B. W. Boehm, "Software Engineering Economics," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 1, pp. 4-21, 1984.
- [35] B. W. Boehm, "Industrial Software Metrics: A Top Ten List," *IEEE Software*, vol. 4, no. 5, pp. 264-271, 1987.
- [36] B. Haskins, J. Stecklein, B. Dick, G. Moroney, R. Lovell and J. Dabney, "Error Cost Escalation Through the Project Life Cycle," *INCOSE International Symposium*, pp. 1723-1737, 2004.
- [37] G. M. Schneider, J. Martin and W. T. Tsai, "An Experimental Study of Fault Detection in User Requirements Documents," *IACM Transactions on Software Engineering and Methodology*, vol. 1, no. 2, pp. 188-204, 1992.
- [38] M. S. Durmuş, İ. Üstüoğlu, R. Y. Tsarev and M. Schwarz, "Modular Fault Diagnosis in Fixed-Block Railway Signaling Systems," *14th IFAC Symposium on Control in Transportation Systems*, pp. 459-464, 2016.



M. S. Durmus (M'07) received the B.S. and M.S. degrees from Pamukkale University (PAU), Turkey, in 2002 and 2005, respectively, and the Ph.D. degree from Istanbul Technical University (ITU) in 2014. From 2007 to 2014, he was a Research and Teaching Assistant with the Department of Control and Automation Engineering, ITU. He is currently a Research and Teaching Assistant with the Department of Electrical and Electronics Engineering, PAU. He also received a second Ph.D. degree from the Division of Electrical, Electronic and Information Engineering, Osaka University, as a Ronpaku fellow of the Japan Society for the Promotion of Science. His research interests include railway signaling systems, functional safety, linear control theory, fault diagnosis of discrete event systems.

Since September
Technical Universi-
systems, railway



Ilker Üstoglu received a B.Sc. degree in electrical engineering from Istanbul Technical University (ITU), Turkey, in 1997 and a M.Sc. degree in control and computer engineering from ITU, in 1999. He completed his Ph.D. in control and automation engineering at ITU in 2009. 2010, he has been working at the Control and Automation Engineering Department of Yıldız University, Turkey. His research areas include linear control theory, computer algebra, fuzzy sets and systems and functional safety.

Roman Yu. Tsarev received the degree of software engineer from the Krasnoyarsk State Technical University, Krasnoyarsk, Russia, in 1999 and a Ph.D. degree in system analysis, control and information processing from Krasnoyarsk State Technical University, Krasnoyarsk, Russia, in 2003. Since then, he has been an Associate Professor with the Department of Informatics, Siberian Federal University, Krasnoyarsk, Russia. He was a recipient of the Award "Honor for Working in Science and Education" from the Russian Academy of Natural History in 2007, the Medal named after A. Nobel for the contribution to the development of inventions from the Russian Academy of Natural History in 2008, the State Prize of the Krasnoyarsk Region for Achievements in Higher Education in 2008, the Gold Medal "For Innovative Work in Higher Education" in 2014. His research interests include N-version programming, software reliability, multiple attribute decision making, automated control systems.



Josef Böröcsök was born in 1959. He received his B.Sc. degree in 1986 (University of Applied Sciences in Darmstadt), the M.Sc. degree in 1991 (University of Kassel) and the Ph. D. in 1995 (Technical University of Ilmenau). He was for more than 10 years researchers in leading positions in the industry (Avionics, Industrial control and Safety-Computer). During this time he lectures at universities, as well as at universities of applied sciences with lectures of automation systems, computer technology, real-time systems, network techniques and safety-related computer technology. Since 2005 he is Professor and head of the Department Computer architecture and System programming at the University of Kassel in Germany. The major research interests are in the field of modelling of safety-related computer technology, real-time systems, network techniques, sensor and distributed sensor systems and calculations models for PFD/PFH and MTTF. He work since years on several national and international standardisations committees and is member of DKE committees, IEEE, IET and BCS. He has more than 200 publications in the refereed international journals.